A Model Comparison Of Layering Effects On Learning In The Cortex and Subcortical Structures

Computational Cognitive Neuroscience Final Term Project

Cognitive Science 123
And
Computer Science and Engineering 173

Written by Team LAAD:
Dylan Amezcua
Leigh Ann Cull
Alivia Harris

Table of Contents

A Model Comparison Of Layering Effects On Learning In The Cortex and Subcortical Structures

Abstract

Sequential learning, with scaling complexity, is an important procedure for learning. This learning process allows one to identify similarities between concepts and build a sense of understanding based upon the shared features. We anticipated the project would result in a prototype computational cognitive neuroscience model, displaying the effects of learning in a multiple layer network, such as those found in the cortex, as compared to a single layer network, such as those found in the subcortex; demonstrating that varying regions of the brain require differentiated learning algorithms and that communication across regions of the brain is mandatory for holistic learning. To do this, we attempted to increase the efficiency of the face categorization neural network, found in the Emergent modeling software, by first adding an additional hidden layer. Later, we attempted to solve the discrepancies seen between the original single layer network and the second hidden layer network by increasing the complexity of the training sequence over time.

Introduction

1.1 Bidirectional networks

Neurons represent relationships between sensory inputs in the world and cognitive responses in the brain. In order to maintain desirable behaviors or brain states, the brain makes use of bidirectional networks. In top-down networks, higher level knowledge (such as *who* someone is) influences how one sorts and processes low level information (such as the color of someone's shirt), usually through a feedback loop. Feedback works by taking activation at the output layer, and working down through the hidden layer, re-adjusting weights based on the original output. Bottom-up networks use feedforward loops, in which information flows "up" the cortical hierarchy, from sensory input to hidden layer, then on to any other additional layers before finally reaching the output. This is the more common way to visualize learning and information processing.

Our model primarily uses feedforward loops, because it looks at the input pattern, which then sends activation up to the hidden network, which organizes the inputs before then sending the activation along to the output, which determines what the network is experiencing. In this model, outputs are represented by identifying specific faces as individuals, and the expressions perceived.

If the model had a top-down mechanic then given the identity and expression of the individual, the network passes the appropriate activation through the hidden layer to try and build a relationship between the locations of the input units. However, our model trains through a bottom-up mechanic. This means given a relationship between locations of the input layer it would pass the correct parts to the hidden layer, which would try to break up the input into the corresponding components, gender and emotion.

1.2 Motivations and Goals

The goal of this model is to compare the effectiveness of layering on learning via bidirectional networks, such as those found in the cortex and subcortical structures of the brain. We anticipate this model will demonstrate that having a hidden layer may have adverse affects on the process of bottom-up learning, due to the increase in transfer points.

The main motivation behind this project was to see if, given certain input patterns, adding additional computational layers were worth the extra space and memory required. This is a common dilemma in most robotics and artificial intelligence.

Additionally, several members of our team are interested in learning from an educational perspective. If a simple network can use less detailed information to still accurately recognize and categorize more difficult patterns, then this provides evidence that students have the ability to discover higher-level concepts given fundamental information, and justifies the push for more individual exploration in classrooms.

Methods

Our approach to this model was to first modify the Face_Categ project from Chapter 3: Networks by training the network on simplified expressions. Second, to add a hidden layer to the functions of the original project.

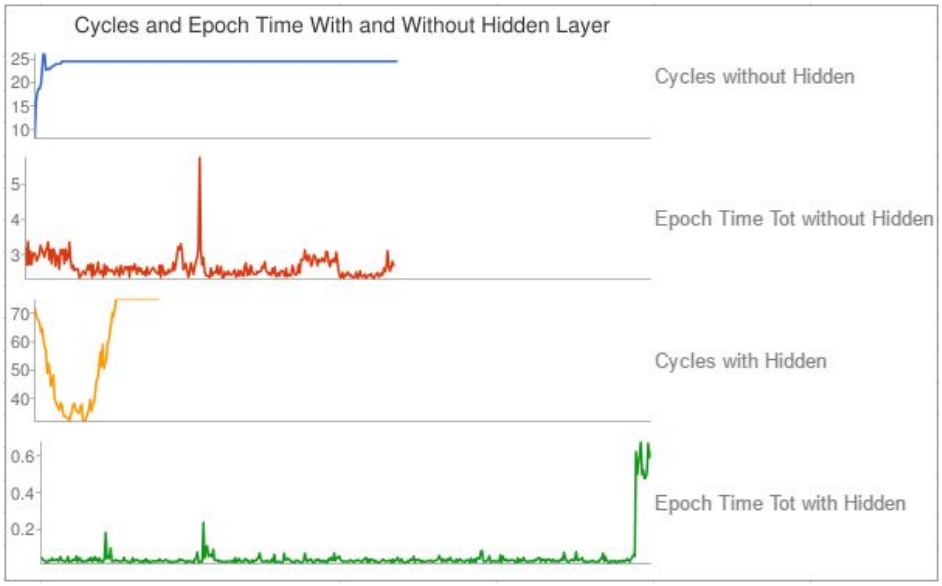
Using a pre-existing lab in the Emergent software called face_categ, we explored how top-down and bottom-up models worked. For bottom-up processing, the network took a set of faces as input and tried to give out the correct output of that face's emotion, gender and identity. The original network was also capable of producing the correct output when trained on a set of partial faces. One could modify the network so that it took a top-down approach after training, and used what was originally an output (emotion, gender, identity) to try to compute what the desired face looked like.

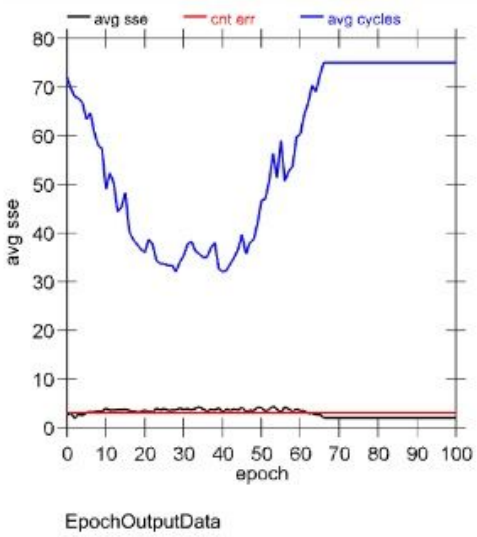
We took this base model and modified the inputs to expand the initial training set, and narrowed down the outputs to produce more focused results. We slightly altered the eyes and nose inputs, then added a flat mouthed to create an additional expression (RB) for all the identities. To compensate, we also extended the range of emotions output to take activation for the RB face. After having added the new faces to our original set of inputs, we tested the network to see if it would accept the changes we did to it. Having the network successfully identify the new expression on that portion of our test, we wanted to test the program against more generic faces.

'Generic' faces were created by taking a copy of one of the original face matrix ('Alberto') for all three expressions and removing everything from it that wasn't his eyes and mouth. We initialized the weights and trained the network on the 'generic' faces until SSE was reported 0, or 41 epochs.

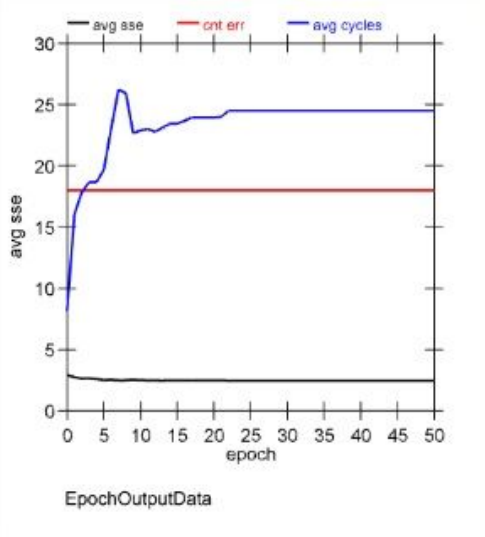
To create a comparison case, we built a second network from the Emergent LeabraStd basic model. To make it more like our original network, the input layer was expanded from a 5X5 to a 16X16 to fit the original input data, and 'identity' and 'gender' target layers were added. Then, the output layer was reduced to 3x1 to improve visual function.

Once the model was complete, the network was trained and tested on the original input data set to ensure the results were consistent with the base model before further modifications. Finally, the hidden layer was constructed in the network. When first running the new multi-layer network with our 'generic' face to train it, the network produced the following unexpected results:



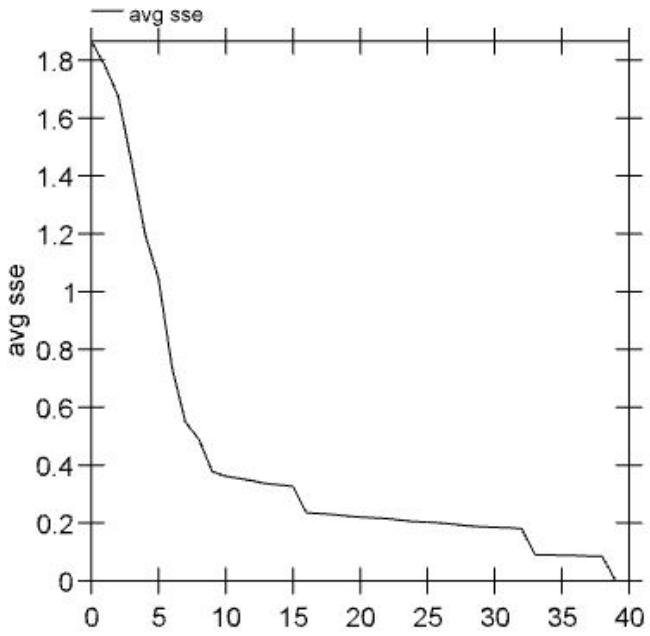


Epoch Table with Hidden Layer

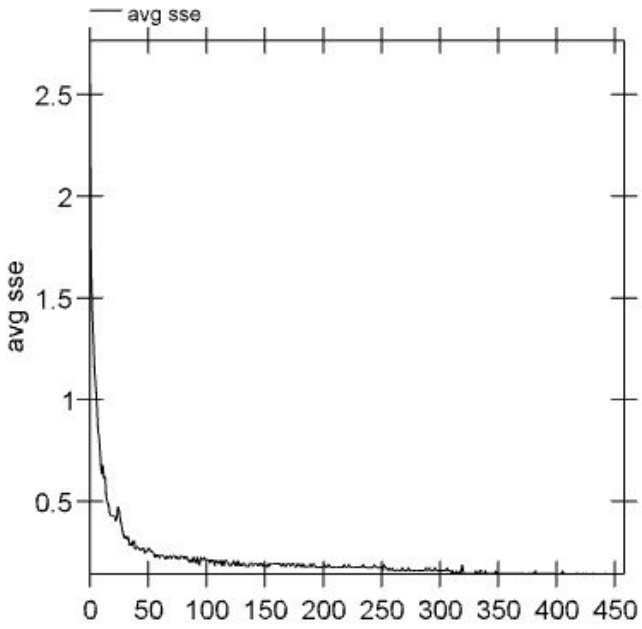


Epoch Table without Hidden Layer

$Figure_1$: Our original graphs



$Figure_2$: The number of errors per epoch during the 'Low' training phase of the single-layer network.



$Figure_2$: The number of errors per epoch during the 'Low-Med' combination training phase of the single-layer network. The network approaches 0 sse but never achieves it during the 500 epochs.

Troubleshooting revealed that the program was not actually learning the input. The issue was resolved after manipulating the learning rate and resetting it back to the default value. The term "sse", which we rely upon heavily for our analysis, stands for 'sum square error,' which is the difference between the target output and the network output.

Once the hidden layer network was outputting as expected, we felt we were getting a good reading off the graphs and decided to build on our network. Similar to the original (hereafter referred to as "high") and generic face (hereafter referred to as "low") data sets, a third data set was created ("medium"). This medium complexity data set was generated in an identical manner to the low complexity data set, with the inclusion of the face shape inputs. This allowed us to compare the two networks on varied initial training conditions.

First, both the single-layer network and the multi-layer network had the weights initialized. Networks were immediately tested on the high complexity faces with no training input to create a control condition. Following, each network was allowed a maximum of 500 epochs to train on the low-complexity input sets; however, both networks reported having learned the data set (SSE error equaled 0) after 41 epochs. Both networks were then presented and tested on the high-complexity faces for an additional 500 epochs, the set standard for all testing procedures. This process was repeated with the medium and high complexity faces, with 500 epochs of training and 500 epochs of testing. The average SSE and average number of cycles were recorded at the end of the testing session.

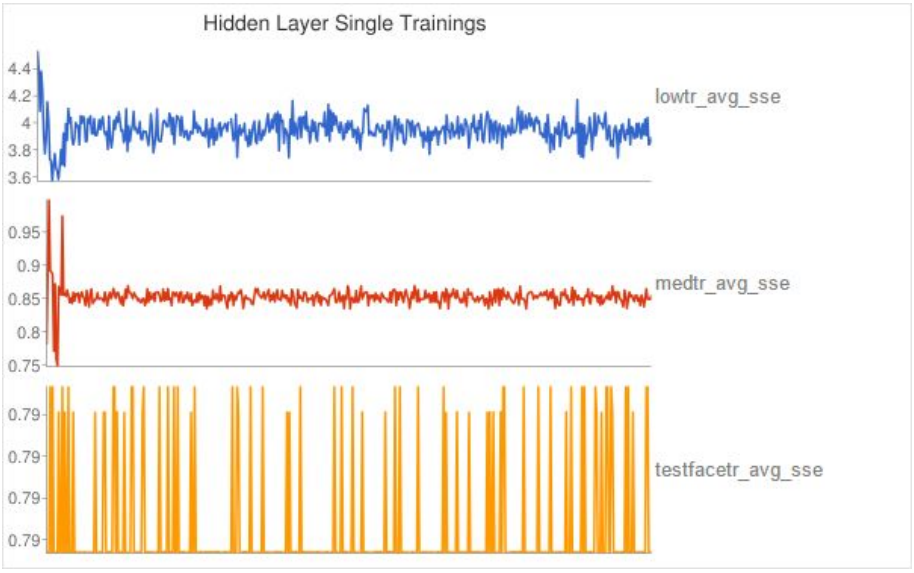
To simulate a "scaffolded" learning, both the single and multi-layer network were trained on multiple data sets in a sequential fashion. The networks were initialized, and trained for 41 epochs on the low data set, then 459 epochs on the medium data set to maintain the consistency of 500 training epochs before being tested. Unfortunately, this consistency could not be maintained in the final trail, as the network struggled to stabilize in the medium and high training conditions. As such, the networks were instead trained on 41 epochs of low data, 250 epochs of medium data, and 250 epochs of high data, and tested on the standard 500 epochs.

Results

$Graph_1$: No hidden layer with single trainings on our "Low" faces that are represented by our generic faces, "Medium" faces that included more facial features, and "TestFace" that included the original faces from the

face_categ.proj with addition to the Resting Faces (RBF) for each identity, and the average SSE for each epoch.

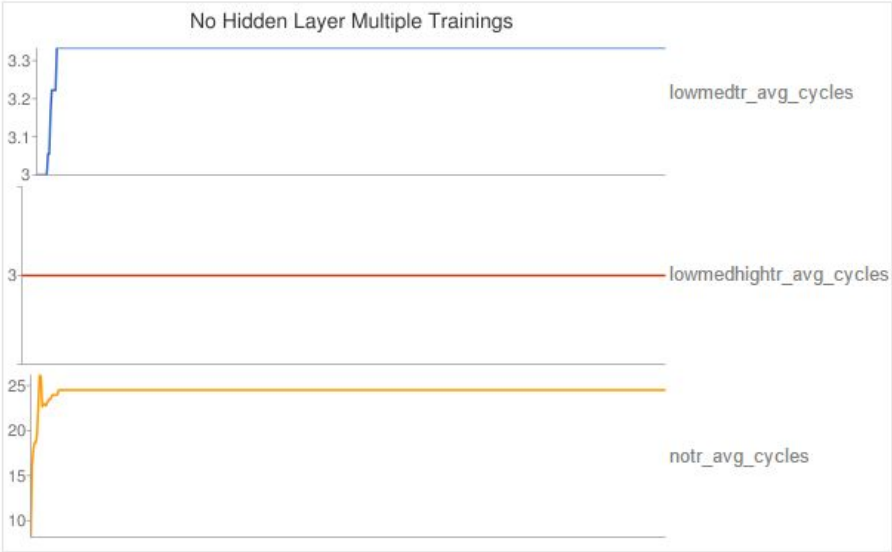
G_2 : No hidden layer with single trainings on our "Low" faces that are represented by our generic faces, "Medium" faces that included noses, and "TestFace" that included the original faces from the face_categ.proj with addition to the Resting Faces (RBF) for each identity, with the average cycle for each epoch.



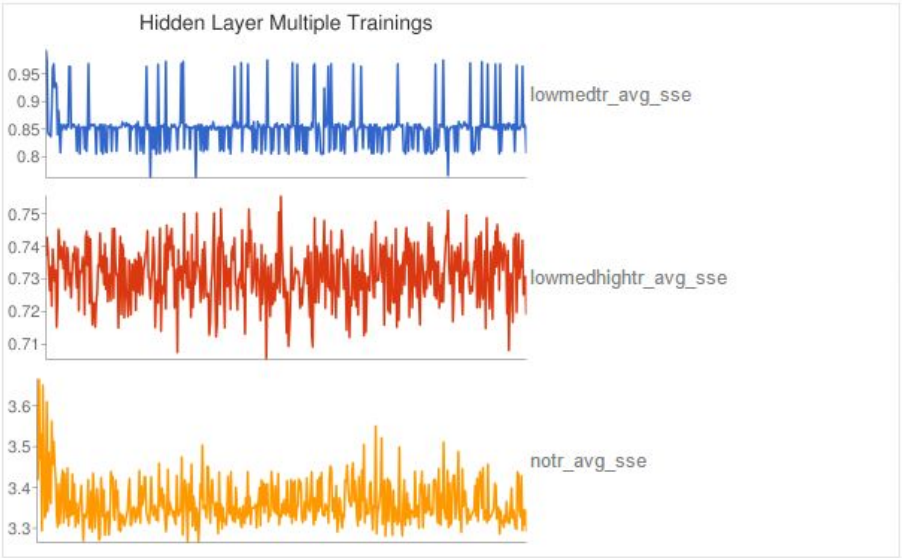
G_3 : Hidden layer with single trainings on our "Low" faces that are represented by our generic faces, "Medium" faces that included noses, and "TestFace" that included the original faces from the face_categ.proj with addition to the Resting Faces (RBF) for each identity, with the average SSE for each epoch.

G_4 : Hidden layer with single trainings on our "Low" faces that are represented by our generic faces, "Medium" faces that included noses, and "TestFace" that included the original faces from the face_categ.proj with addition to the Resting Faces (RBF) for each identity, with the average cycle for each epoch.

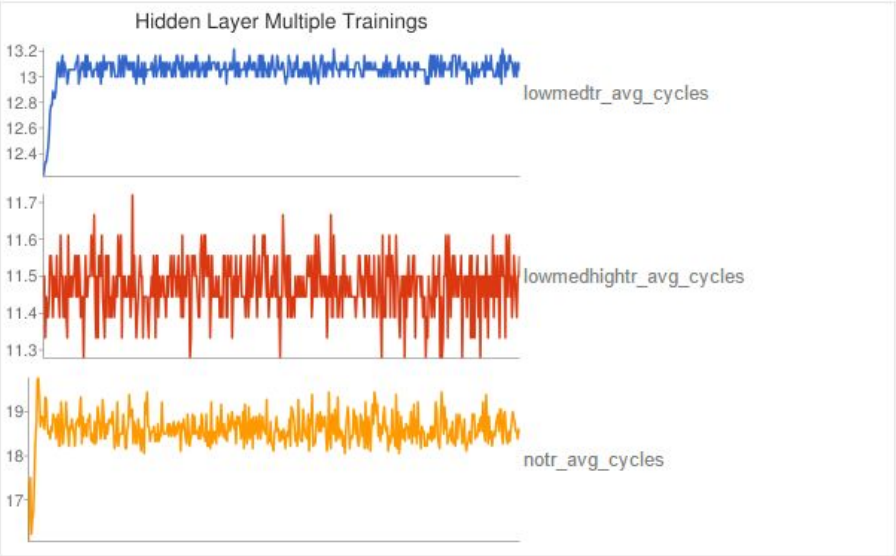
G_5 : No hidden layer with multiple trainings first on our "Low" faces that are represented by our generic faces and "Medium" faces that included noses. The "Low" faces and the "Medium" faces followed by the "High" which used the same faces from our"TestFace" that included the original faces from the face_categ.proj with addition to the Resting Faces (RBF) for each identity. And finally no training what so ever before testing. Each is shown with the average SSE for each epoch.



G_6 : No hidden layer with multiple trainings first on our "Low" faces that are represented by our generic faces and "Medium" faces that included noses. The "Low" faces and the "Medium" faces followed by the "High" which used the same faces from our"TestFace" that included the original faces from the face_categ.proj with addition to the Resting Faces (RBF) for each identity. And finally no training what so ever before testing. Each is shown with the average cycle for each epoch.



G_7 : Hidden layer with multiple trainings first on our "Low" faces that are represented by our generic faces and "Medium" faces that included noses. The "Low" faces and the "Medium" faces followed by the "High" which used the same faces from our"TestFace" that included the original faces from the face_categ.proj with addition to the Resting Faces (RBF) for each identity. And finally no training what so ever before testing. Each is shown with the average SSE for each epoch.



G_8 : Hidden layer with multiple trainings first on our "Low" faces that are represented by our generic faces and "Medium" faces that included noses. The "Low" faces and the "Medium" faces followed by the "High" which used the same faces from our"TestFace" that included the original faces from the face_categ.proj with addition to the Resting Faces (RBF) for each identity. And finally no training what so ever before testing. Each is shown with the average cycle for each epoch.

On average the single network made less errors (avg_sse = 2.26) compared with the hidden layer (avg_sse = 3.93) during our "Single Training" tests on the "Low" faces, although it did take less time for the hidden layer network to initially learn (the avg_cycles being 18.94 compared to the 35.96 for the single layer network).

Actually, when comparing the average SSE and average cycles for each trial, the single layer network seemed to outperform the hidden layer network on every trial except for two. The one mentioned before on "Low" training faces, and on the multiple training trial when we trained the network initially on the "Low" faces and then immediately on the "Medium" faces. During this trial the hidden layer network scored an average SSE of only 0.85 and had an average cycle of 13.05 compared to the single layer network's average of 2.46 and 24.38 respectively.

Discussion

Our results were as we expected, that the single layer would reach the desired output faster than the one with the hidden unit, but we did not expect that the single layer would have less errors as well. We anticipated the single layer to be faster because it goes directly to the output without having to excite another level of neurons, but we assumed having the hidden layer would help minimize the amount of errors. With the large selection of testing we did with our network, it was surprising that the single layer was only beaten once, in the low-mid training condition.

If we were to do an extension on this topic we would first expand the training to include training on generic/high faces, and medium/high faces. Second, see how changing the size of the hidden unit would change the network before seeing how two or more hidden layers would affect the network, and whether it would slow it down a lot more compared to the single or if it would end up becoming more efficient.

This model may also be extended to investigate the breakdown of information in autistic individuals. A symptom seen throughout the autistic spectrum is difficulty recognizing emotions, or failing to interpret more abstract ideas. The issue does not seem to be in the processing of the input data, but rather a failure or mishandling on the part of the weights between the input and target layers.

Conclusion

After multiple training and testing phases of our neural network models, it may be concluded that if one wants to quickly recognize and organize patterns, it may be beneficial to make sure the input stays in local memory and does not go needlessly transverse through hidden networks.

When new training input samples are added, the network continues to learn, but at an insignificant rate. Scaffolding learning seemed much more beneficial to the multi-layer network's learning than the single layer network. The network produces a high amount of initial errors, and only shows true 'learning' when presented with only one form of input data. Otherwise, the network does not settled down into a zero-error state. Later, when tested on the higher complexity data, it is only natural that the instances where the network is trained on the same high complexity data, it performs the best. It is interesting to note, however, that is does not perform perfectly.

To conclude the goal of our network proved what we believed that adding a hidden layer would cause adverse effects to learning. This would cause wasted space in memory and slow down the results if used in robotics/ A.I. The model can be expanded in the future by taking into consideration more than one base for multiple data sets, and verified by running it in other neural network simulators, such as NetLogo or MATLAB. Moreover, we can further extend the model by expanding the number and size of the hidden layers, or expanding the training database.

Acknowledgements

Thanks to our teammates for being wonderfully helpful, patient, and awesome! Dr. David Noelle for his support and teachings this semester. Jeffrey Rodney and Jacob Rafati for their help (with breaking our sim) and support. Most of all, for that beautiful and wonderful save button.

Should one have any other questions regarding our program and/or results please feel free to email one of our team members at any of the following email addresses.

Dylan Amezcua: damezcua2@ucmerced.edu

Leigh Ann Cull: lcull@ucmerced.edu

Alivia Harris: aharris38@ucmerced.edu

References

Frank, Michael, Yuko Munakata, Thomas Hazy, Randall O'Reilly, and Contributors. "CCNBook/Learning." CCNBook/Learning - Computational Cognitive Neuroscience Wiki. N.p., n.d. Web. 08 May 2017.

Frank, Michael, Yuko Munakata, Thomas Hazy, Randall O'Reilly, and Contributors. "CCNBook/Networks." CCNBook/Networks - Computational Cognitive Neuroscience Wiki. N.p., n.d. Web. 08 May 2017.

Frank, Michael, Yuko Munakata, Thomas Hazy, Randall O'Reilly, and Contributors. "CCNBook/Sims/All." CCNBook/Sims/All - Computational Cognitive Neuroscience Wiki. N.p., n.d. Web. 08 May 2017.

O'Reilly, Randall C. "CCNBook/Sims/Networks/Categ." CCNBook/Sims/Networks/Categ - Computational Cognitive Neuroscience Wiki. O'Reilly, Munakata, Frank, Hazy, & Contributors, 27 July 2016. Web. 08 May 2017.